Étape 10 : Déplacement sensoriel, partie 1.

Jamila Sam & Jean-Cédric Chappelier, 2018

Version : 1.2

But

Modéliser et commencer à intégrer les phéromones.

Description

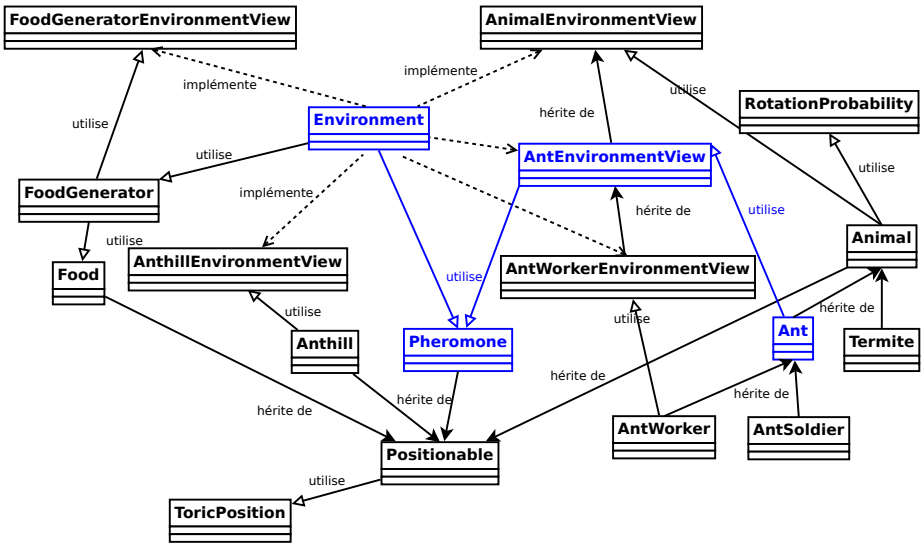

Figure 1: Composants impliqués dans l'étape 10 (en noir les composants déjà codés, en vert les nouveaux composants, en bleu ceux retouchés)

Les fourmis se déplaçant de façon aléatoire, elles ont peu de chance de retrouver leur fourmilière pour, par exemple, y déposer la nourriture récoltée.

Nous allons à cette étape simuler le déplacement sensoriel des fourmis. Ces dernières vont utiliser des traces de phéromone laissées par elles-mêmes ou leur

congénères, pour retrouver leur chemin.

La mise en place du déplacement sensoriel est assez conséquente et fera l'objet de trois étapes du projet. Les deux premières sont consacrées à la mise en place des éléments nécessaires au déplacement sensoriel. La troisième étape fera la synthèse pour assembler le tout.

Dans cette première partie, nous allons nous occuper des phéromones, de leur dispersion et leur repérage par les fourmis. La figure 1 montre les composants impliqués par cette étape.

Implémentation

Phéromones

Une classe `Pheromone` sert à modéliser les traces de phéromone. Une instance de cette classe sera caractérisée par une quantité (un `double`), ainsi que d'une position (torique) héritée de `Positionable`.

Les autres méthodes qu'il vous faudra implémenter sont :

- le constructeur initialisant la position et la quantité au moyen de valeurs passées en paramètres ;
- la méthode `double getQuantity()` retournant tout simplement la quantité de phéromone ;
- une méthode `boolean isNegligible()` retournant `true` si la quantité de phéromone est considérée comme négligeable (c.-à-d. concrètement, inférieure strictement à la constante `PHEROMONE_THRESHOLD`) ;
- une méthode `void update(Time dt)` faisant évoluer la quantité de phéromone au cours du temps. La quantité de phéromone s'évaporera selon l'algorithme suivant : à chaque pas de temps, si elle est non négligeable, elle sera décrémentée du pas de temps (`dt.toSeconds()`) multiplié par le taux d'évaporation `PHEROMONE_EVAPORATION_RATE`. La quantité sera ramenée à zéro si elle devait devenir négative après ce calcul.

Intégration à l'environnement

Nous devrions pouvoir être capable d'ajouter des instances de `Pheromone` dans l'environnement. Ajoutez pour cela la méthode `void addPheromone(Pheromone pheromone)` dans l'interface `AntEnvironmentView`. La classe `Environment` devra implémenter cette méthode ; comme son nom l'indique, elle devra ajouter des phéromones dans une `LinkedList` de `Pheromone`. Si l'argument

passé à cette méthode vaut `null`, la méthode `addPheromone` lancera une `IllegalArgumentException`.

Ajoutez également la méthode publique `List<Double> getPheromonesQuantities()` dans `Environment` qui retourne simplement la liste des quantités de phéromone présentes dans l'environnement.

Dans la méthode `update()`, vous devrez parcourir la liste des phéromones pour retirer celles qui sont négligeables et pour faire évoluer les autres en appelant leur méthode `update()` (inspirez vous de ce que vous avez fait pour l'évolution des animaux).

L'ordre des méthodes de mises à jour est le suivant :

- génération de nourriture ;
- gestion des phéromones ;
- gestion des animaux (génération puis mise à jour) ;
- nettoyage des instances de nourriture avec une quantité nulle.

Les fourmis déposent des traces de phéromone

Les fourmis doivent déposer des traces de phéromone en se déplaçant.

Supposons que la dernière trace de phéromone ait été déposée à la position `lastPos` et que la fourmi se soit déplacée jusqu'à la position `currentPos`.

L'algorithme déposant des traces de phéromone aura pour rôle de créer `d * densite` instances de `Pheromone`, réparties à distances égales, sur le chemin entre `lastPos` et `currentPos` (inclut). `d` est la distance torique entre `lastPos` et `currentPos`, et `densite` est la densité de phéromone, donnée par la constante `ANT_PHEROMONE_DENSITY`.

Créez une méthode privée

```
void spreadPheromones(AntEnvironmentView env)
```

dans la classe `Ant` réalisant ce traitement. Vous supposerez que `lastPos` correspond à la position de la fourmi à sa création.

Comme la dispersion des phéromones restera dans le cadre de ce projet identique pour tous les types de fourmis, elle ne devra pas pouvoir être redéfinie.

Pour le moment, cette méthode ne sera pas utilisée (mais voir les suggestions de test plus bas), nous allons l'intégrer dans la partie suivante.

Remarques

- Pensez aux méthodes `toricVector()` et `toricDistance()` que vous avez codées dans la classe `ToricPosition`.

- Chaque trace de phéromone sera créée avec une quantité `ANT_PHEROMONE_ENERGY`.

Repérage de la phéromone entourant la fourmi

Nous allons maintenant implémenter le repérage des quantités de phéromone disponibles autour de la fourmi. Le voisinage de la fourmi est découpé en secteurs d'angles, selon un tableau d'angles comme celui-ci :

```
(   -180,   -100,   -55,   -25,   -10,    0,
      10,     25,    55,   100,   180        )
```

(par rapport à la fourmi ; l'angle 0 indique donc « devant la fourmi »).

Il s'agit de trouver les quantités de phéromone présentes au plus proche de chacun de ces angles d'observation, mais dans la limite des capacités sensorielles de la fourmi. On considérera qu'une fourmi ne peut sentir que ce qui est à moins de `ANT_SMELL_MAX_DISTANCE` d'elle (rayon olfactif de la fourmi).

Il vous est demandé d'ajouter à l'interface `AntEnvironmentView` (et de l'implémenter dans `Environment`), la méthode

```
double[] getPheromoneQuantitiesPerIntervalForAnt(ToricPosition position,
                                    double directionAngleRad, double[] angles)
```

prenant comme paramètre la position de la fourmi, son angle de direction, et un tableau d'angles (comme celui montré ci-dessus).

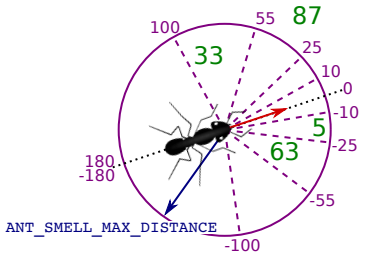

Figure 2: Un exemple de perception des phéromones (indiquées en vert). Voir le texte principal pour plus d'explications.

La méthode retourne le tableau des quantités de phéromone perçues dans chaque secteur d'angle.
Par exemple, pour la situation illustrée en figure 1 (correspondant au tableau d'angles donné ci-dessus) et pour laquelle il y a 63 de phéromone à un angle de -35 degrés par rapport à la direction de la fourmi, 5 de phéromone à un angle de -20 degrés, 87 à un angle de 40 degrés et 33 à un angle de 90 degrés, le tableau retourné serait :

```
( 0, 0, 0, 68, 0, 0, 0, 0, 0, 33, 0 )
```

car il n'y a aucune phéromone entre -180 et -100 degrés par rapport à la direction de la fourmi, il y a 63 de phéromone à -35 degrés dont l'angle le plus proche parmi les angle d'observation retenus est -25 degrés, 5 à -20 degrés dont l'angle d'observation le plus proche est à nouveau -25 degrés ; ce 5 s'additionne donc au 63 précédent ; puis enfin 33 de phéromone au plus proche des 100 degrés. Les 87 de phéromone situés à 40 degrés ne sont pas comptabilisés car non perçus par la fourmi : ils sont à une distance supérieure à `ANT_SMELL_MAX_DISTANCE` de celle-ci.

Un algorithme simple pour calculer le tableau `T` à retourner est le suivant :

- pour toute (instance de) phéromone `p` non négligeable et qui est dans le rayon olfactif de la fourmi :
  - trouver l'angle `beta` que fait cette phéromone avec la direction de la fourmi ;
  - trouver l'angle parmi les angles d'observation retenus duquel `beta` est le plus proche ; ajoutez la quantité de `p` à `T[i]` avec `i` l'index de l'angle trouvé.

Pour trouver l'angle `beta` : calculer le vecteur torique `v` entre la position de la fourmi et celle de la phéromone. L'angle beta vaut simplement `v.angle() - directionAngleRad` où `directionAngleRad` est l'angle de direction de la fourmi.

Trouver l'angle le plus proche est un peu plus subtil. Pour cela, il vous est conseillé d'écrire les deux méthodes statiques privées suivantes :

- `double normalizedAngle(double angle)` qui normalise l'angle, c.-à-d. qu'on lui ajoute 2π tant qu'il est inférieur à zéro et qu'on lui retranche 2π tant qu'il est supérieur à 2π.
- `double closestAngleFrom(double angle, double target)` qui traite le problème mentionné ci-dessus. L'idée est la suivante : on calcule l'angle `diff = angle - target` qu'on normalise, puis on prend le minimum entre ce `diff` et `2*Math.PI - diff`.

Ainsi, `closestAngleFrom()` s'appuiera sur `normalizedAngle()`. De même, `getPheromoneQuantitiesPerIntervalForAnt()` s'appuiera sur `closestAngleFrom()`. On appellera `closestAngleFrom()` avec comme paramètre `angles[i]` pour `angle` et `beta` pour `target`.

Tests et soumission

Tests locaux

Nous vous encourageons à ajouter vos propres tests ponctuels dans le fichier `ch/epfl/moocprog/tests/Main.java` pour tester votre code avant de le soumettre.

Voici un exemple de code que vous pourriez ajouter à ce fichier :

```
import static ch.epfl.moocprog.config.Config.PHEROMONE_THRESHOLD;
import ch.epfl.moocprog.Pheromone;

//....

// quelques tests pour l'étape 10
System.out.println();
double minQty = getConfig().getDouble(PHEROMONE_THRESHOLD);
Pheromone pher1 = new Pheromone( new ToricPosition(10.,10.), minQty);
System.out.print("Pheromone pher1 created with quantity PHEROMONE_THRESHOLD = ");
System.out.println( minQty );
System.out.println("the position of the pheromone is :"  + pher1.getPosition());
System.out.println("getQuantity() correctly returns the value " + minQty + " : "
     + (pher1.getQuantity() == minQty));
System.out.print("the quantity of the pheromone is negligible : ");
System.out.println(pher1.isNegligible());

env = new Environment();
env.addPheromone(pher1);
env.update(Time.fromSeconds(1.));
System.out.print("After one step of evaporation (dt = 1 sec), ");
System.out.print(" the quantity of pher1 is ");
System.out.println(pher1.getQuantity() + "\n");

double offset = minQty / 5.;
Pheromone pher2 = new Pheromone( new ToricPosition(20.,20.),
             getConfig().getDouble(PHEROMONE_THRESHOLD) - offset);
System.out.println("Pheromone created with quantity PHEROMONE_THRESHOLD - " + offset);
System.out.println("the position of the pheromone is :"  + pher2.getPosition());
System.out.print("the quantity of the pheromone is negligible : ");
System.out.println(pher2.isNegligible()  + "\n");
env.addPheromone(pher2);

System.out.print("The quantities of pheromone in the environment are: ");
System.out.println(env.getPheromonesQuantities());
```

```
env.update(Time.fromSeconds(1.));
// toute les quantités deviennent négligeables et doivent être supprimées
System.out.print("After one update of the environment, ");
System.out.print("the quantities of pheromone in the environment are: ");
System.out.println(env.getPheromonesQuantities() + "\n");

System.out.println("Finding pheromones around a given position : ");
ToricPosition antPosition = new ToricPosition(100., 100.);
Pheromone pher3 = new Pheromone(new ToricPosition(105., 105.), 1.0);
Pheromone pher4 = new Pheromone(new ToricPosition(95., 95.), 2.0);
// cette quantité est trop éloignée (ne doit pas être perçue) :
Pheromone pher5 = new Pheromone(new ToricPosition(500., 500.), 4.0);
env.addPheromone(pher3);
env.addPheromone(pher4);
env.addPheromone(pher5);
System.out.print("The quantities of pheromone in the environment are: ");
System.out.println(env.getPheromonesQuantities());
double[] pheromonesAroundPosition =
     env.getPheromoneQuantitiesPerIntervalForAnt(antPosition, 0.,
             new double[] {Math.toRadians(-180), Math.toRadians(-100),
                           Math.toRadians(-55),  Math.toRadians(-25),
                           Math.toRadians(-10),  Math.toRadians(0),
                           Math.toRadians(10),   Math.toRadians(25),
                           Math.toRadians(55),   Math.toRadians(100),
                           Math.toRadians(180)
                       });
System.out.println(Arrays.toString(pheromonesAroundPosition) + "\n");
System.out.print("After enough time, no pheromones should be left : ");
env.update(Time.fromSeconds(30.));
System.out.println(env.getPheromonesQuantities());
```

Ce qui devrait produire les affichages suivants :

```
Pheromone pher1 created with quantity PHEROMONE_THRESHOLD = 0.01
the position of the pheromone is :Position : 10.0, 10.0
getQuantity() correctly returns the value 0.01 : true
the quantity of the pheromone is negligible : false
After one step of evaporation (dt = 1 sec), the quantity of pher1 is 0.0

Pheromone created with quantity PHEROMONE_THRESHOLD - 0.002
the position of the pheromone is :Position : 20.0, 20.0
the quantity of the pheromone is negligible : true

The quantities of pheromone in the environment are: [0.0, 0.008]
After one update of the environment, the quantities of pheromone in the
environment are: []
```

```
Finding pheromones around a given position :
The quantities of pheromone in the environment are: [1.0, 2.0, 4.0]
[0.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0]

After enough time, no pheromones should be left : [0.0, 0.0, 0.0]
```

Libre à vous d'étoffer et compléter ces tests selon ce qui vous semble pertinent.

Concernant la méthode `spreadPheromone()`, comme celle-ci n'est pas encore intégrée à la simulation à proprement parler (il vous manque encore quelques outils), il n'est pas possible de constater visuellement le dépôt des phéromones. Ce sera par contre possible dès la prochaine étape. Cette méthode étant par ailleurs exigée comme `private` par le correcteur automatique, vous ne pouvez pas non plus la tester dans vos tests locaux en l'état. Nous vous conseillons pour cela, ponctuellement (en attendant la prochaine étape), de la mettre en `public` et faire quelques tests locaux, plus la remettre en `private` avant de soumettre votre code au correcteur automatique.

Voici un exemple simple de test que vous pourriez alors faire :

```
AntWorker antWorker = new AntWorker(new ToricPosition(), Uid.createUid());
env.addAnimal(antWorker);
env.update(Time.fromSeconds(0.5));
antWorker.spreadPheromones(env);
System.out.println("The antworker moved. Distance : "
                   + new ToricPosition().toricDistance(antWorker.getPosition()));
double density = getConfig().getDouble(ANT_PHEROMONE_DENSITY);
double energy = getConfig().getDouble(ANT_PHEROMONE_ENERGY);
System.out.println("The density of pheromone is : "
                   + density + " and the energy : " + energy);
System.out.println("The antworker spreads pheromones");
System.out.println("The quantities of pheromone in the environment are now: ");
System.out.println(env.getPheromonesQuantities());
```

Ce qui devrait produire la sortie suivante :

```
The antworker moved. Distance : 50.0
The density of pheromone is : 0.1 and the energy : 5.0
The antworker spreads pheromones, the quantities of pheromone in the environment
are now: [5.0, 5.0, 5.0, 5.0, 5.0]
```

Le code permettant le test de `spreadPheromones` serait donc bien sûr à commenter dès que la méthode `spreadPheromones` sera redevenue `private`, comme prévu initialement.

Soumission

Pour soumettre votre devoir au correcteur automatique, il faut créer un fichier ZIP contenant **tous** vos fichiers sources personnels, depuis la racine `ch/` ; ceux de cette étape, mais aussi ceux de l'étape précédente. Concrètement, pour ce devoir ci, votre fichier ZIP doit donc contenir exactement les fichiers suivants :

```
ch/epfl/moocprog/AnimalEnvironmentView.java
ch/epfl/moocprog/Animal.java
ch/epfl/moocprog/AntEnvironmentView.java
ch/epfl/moocprog/AnthillEnvironmentView.java
ch/epfl/moocprog/Anthill.java
ch/epfl/moocprog/Ant.java
ch/epfl/moocprog/AntSoldier.java
ch/epfl/moocprog/AntWorkerEnvironmentView.java
ch/epfl/moocprog/AntWorker.java
ch/epfl/moocprog/Environment.java
ch/epfl/moocprog/FoodGeneratorEnvironmentView.java
ch/epfl/moocprog/FoodGenerator.java
ch/epfl/moocprog/Food.java
ch/epfl/moocprog/Pheromone.java
ch/epfl/moocprog/Positionable.java
ch/epfl/moocprog/RotationProbability.java
ch/epfl/moocprog/Termite.java
ch/epfl/moocprog/ToricPosition.java
```

Note : si c'est plus pratique pour vous, vous *pouvez* aussi faire un zip qui contient tout le sous-répertoire `ch/epfl/moocprog`, y compris, donc, les fichiers que nous vous avons fournis. Ces fichiers seront simplement ignorés par le correcteur automatique (et remplacés par les nôtres). L'énoncé « *Procédure de soumission* » de la semaine 1 vous indique comment procéder. **Avant de passer à l'étape suivante, n'oubliez pas de faire une sauvegarde de l'étape en cours**, comme indiqué dans le même document.